Java Reviews

Java Review:

type := primitiveType | className | enumTypeName | abstractClassName | interfaceName

primitiveType := boolean | byte | char | double | float | int | long | short

数据类型	大小(bit)	范围	默认值
byte(字节)	8	-128 ~ 127	0
short(短整型)	16	-32768 ~ 32767	0
int(整形)	32	-2147483648 ~ 2147483647	0
long(长整型)	64	-9233372036854477808 ~ 9233372036854477807	0L
float(浮点型)	32	-3.40292347E+38 ~ 3.40292347E+38	0.0f
double(双精度)	64	-1.79769313486231570E+308 ~ 1.79769313486231570E+308	0.0d
char(字符)	16	'\u0000' ~ 'u\FFFF'	'\u0000'
boolean(布尔型)	1	true / false	false

Type	Valid promotions
`double`	None
`float`	`double`
`long`	`float` or `double`
`int`	`long`, `float` or `double`
`char`	`int`, `long`, `float` or `double`
`short`	`int`, `long`, `float` or `double` (but not `char`)
`byte`	`short`, `int`, `long`, `float` or `double` (but not `char`)
`boolean`	None (`boolean` values are not considered to be numbers in Java)

Fig. 6.4 | Promotions allowed for primitive types.

```
Variables, Constants and Values:

modifiers type variableName;
modifiers type variableName = expression;   // initial value

modifiers := accessModifier + static | 无(non-static)
accessModifier = public | protected | 无(package) | private

modifiers final type CONSTANT_NAME;
CONSTANT_NAME = expression;  // only in constructor for final instance variable

modifiers final type CONSTANT_NAME = constantExpression;


Local variable: in method or block or parameterList   // No Default value!!!

Instance variable has a default value after constructing.

Static variable (class variable) has a default value after loading into JVM.

All non-local variable with a reference type has a default value of null.
```

Selected Operators and Their Precedence

(See Appendix A for the complete list.)

`[]`	Array element access
`++ -- !`	Increment, decrement, Boolean *not*
`* / %`	Multiplication, division, remainder
`+ -`	Addition, subtraction
`< <= > >=`	Comparisons
`== !=`	Equal, not equal
`&&`	Boolean *and*
`||`	Boolean *or*
`=`	Assignment

+ - * / 是重载(overloading)操作符，可用于不同数据类型，% 求整数除法的余数(也称取模)
这些二元运算都是从左到右运算的：即 e1 op e2 都是先算 e1，再算 e2，最后算 e1 op e2

++n 与 n++ 的区别：

```
if n = 8, ++n ==> 9 and n is 9
if n = 8, n++ ==> 8 and n is 9
if n = 8, ++n + n ==> 18 and n is 9
if n = 8, n++ + n ==> 17 and n is 9
```

c ? e1 : e2 是条件运算符表达式，? : 为条件运算符

a + b >= c && m * n < p 会自动按动算符优先级运算，不需要加 () 指定运算顺序

```
method Declaration

accessModifier (static | 无 | final) returnType methodName (parameterList) {
    ...

}

accessModifier := public | protected | 无(package) | private
accessModifier严格限制访问(可见)范围。

parameterList := type1 v1, type2 v2, ..., typen vn

static 表明是静态方法(类方法), 无static表明是实例方法(对象方法)
加 final 表明此方法不能在子类中重写(override)

returnType 表明方法的返回值的类型(用 return expression; ), 为 void 时表明无返回值

可以在一个类中定义相同名称的方法(称为方法的重载overloading, 注意与override区分)
同名的方法以参数个数/参数类型的差异确定调用具体那一个方法。


特别的类方法:
public static void main (String[] args) { ... }
Java 程序的入口(启动)方法。
```

```
class Declaration

accessModifier class className
+ extends superClassName
+ implements interfaceNameA, interfaceNameB
{
    static variable declarations (with/without initialization)
    static constant declarations (with initialization)
    static method declarations

    static (class) initial blocks (code)

    instance variable declarations (with/without initialization)
    instance constant declarations (normaly without initialization)
    constructor declarations (this(), super() as the first statement if needed)
    instance method declarations (may overwrite method)

    instance initial blocks (code)

    inner type declarations
}
```

一个java文件里只能定义一个 public class! (可以定义多个非 public class)

```java
public class MySubClass extends TheSuperClass {

   public MySubClass () { // 如不定义无参数构方法，JVM会自动生成一个无参数构造方法
      super(); // 如果无明确调用superClass的构造方法，会自动调用超类无参数构造方法
      ...
   }

   // 可定义自己的方法
   // 也可重写(overwrite)superClass的方法
   // 可定义自己的instance varibales
   // 但不能访问superClass任何private声明的变量和方法
}

TheSuperClass v1 = new TheSuperClass;    // ok
TheSuperClass v2 = new MySubClass();     // ok
MySubClass v3 = new MySubClass();        // ok

v1 = v3;  // ok, Type Promotion
v3 = v2;  // wrong!
v3 = (MySubClass) v2;    // ok , Type Casting
v3 = (MySubClass) v1;    // wrong!
```

```java
public abstract class AnAbstractClass {
   // No Constructor
   public abstract SomeType abstractMethodA (Type1 v1...);  // 抽象方法只声明，无定义
   public SomeType methodB (Typex vx...) { ... }  // 非抽象方法有声明有定义
   ...
}

public class MyClass extends AnAbstractClass {

   public SomeType abstractMethodA (Type1 v1...) { ... }  // 定义超类中的抽象方法
   public SomeType methodB (Typex vx...) { ... }  // 重写(overwrite)超类的方法
                                                  // 重写的方法可见性不能收窄
     // 可定义自己的方法或实例变量
}

AnAbstractClass v1 = new AnAbstractClass();  // wrong, 抽象类不能实例化
AnAbstractClass v2 = new MyClass();   // ok, 可实现多态
MyClass v3 = new MyClass();  // ok
```

```java
public interface TheInterface {
   type1 methodA (Typex v1...);  // 缺省 public abstract
   type2 methodB (Typey p1...);  //
   ...
}

// Java 的 interface 即是纯抽象类，可以implements多个接口，(只能实现一个类)

public class MyClass implements TheInterface {
   public type1 methodA (Typex v1...) { ... }  // 定义接口中声明的抽象方法，+ public
   public type2 methodB (Typey p1...) { ... }  // 定义(重写)的方法可见性不能收窄
   ...
     // 可定义自己的方法或实例变量
}

TheInterface v1 = new TheInterface();  // wrong, 接口不能实例化
TheInterface v1 = new TheInterface() { ...接口方法实现代码... };    // ok, 匿名对象
TheInterface v2 = new MyClass();   // ok, 可实现多态
MyClass v3 = new MyClass();  // ok
```

```
public enum Suit { SPADE, HEART, DIAMOND, CLUB };


Suit s1 = Suit.HEART;


if (s1 != Suit.CLUB) {
    ...
}

switch (sx) {
    case Suit.SPADE: ... break;
    case Suit.HEART:
    case Suit.DIAMOND: ... break;
    case Suit.CLUB: ... break;
}
```

1) 0+ Loop (while repetition)

Syntax:

```
while (condition)
    statement                  // ;  is one part of a simple statement
```

Normally:

```
initialization;
while (condition) {
    statements
}
```

2) 1+ Loop (do-while repetition)

Syntax:

```
do
   statement                    // ;  is one part of a simple statement
while (condition);
```

Normally:

```
initialization;
do {
   statements
} while (condition);
```

3) Stepwise Increment Loop (for repetition)

Syntax:

```
for (initialization; condition; increments)
    statement          // ; is one part of a simple statement
// initialization like:  int  i = 0    or   a = xxx, b = nnn, c = yyyy
// increments like:  i++   or   i++, a = xxx, b = nnn, c = yyyy
```

Normally:

```
initialization;
for (initialization; condition; increments) {
  statements
}
```

4) for each Loop (Enhanced for repetition)

Syntax:

```
for (type element : collection)
    statement          // ; is one part of a simple statement
// for each element in the collection do the statement
// collection can be an array or ArrayList or other Collections
```

Normally:

```
initialization;
for (type element : collection) {
    statements
}
```

5) ½ + Loop (infinite repetition with break)

```
statementsA
while (condition) {
    statementsB
    statementsA
}
```

```
while (true) {    // for (;;)
    statementsA
    if (!condition) break;
    statementsB
}
```

1） 请写出典型switch 语句的语法格式；

```
switch (integrelExpression) {     // integerals, String, enum type
   case c1:       // a constant
      ..
      break;      // might be falling down
   case c2:
      ..
      break;
   ..
   case cn:
      ..
      break;
   default:
      ..
      break;
}
```

2） 请写出典型的 try-catch-finally 结构的语法格式；

```
try {
    ..      // Statements may throw Exceptions
} catch (ExceptionType1 e1) {
    ..      // Deal with Exception Type 1
} catch (ExceptionType2 e2) {
    ..
} catch (ExceptionTypen en) {
    ..
} finally {
   ..       // close & free resources
}
```

3) 请使用条件表达式 (即：c ? e1 : e2)写出与下述代码等效的代码：

```
if (x + y > z)
    a = x + y;
else
    a = z;
```

```
(x + y > z) ? a = x + y : a = z;
```

```
a = (x + y > z) ? x + y : z;
```

4) 请写出与下述代码等效的尽可能简单的代码：

```
// isConfirmed()是返回boolean类型的一个方法
if (isConfirmed())
   return false;
else
   return true;
```

```
isConfirmed() ? return false : return true;
```

```
return isConfirmed() ? false : true;
```

```
return ! isConfirmed();
```